Interactive Engineering Calculation Sheets with Org-Mode and Python

P. Vennemann

February 8, 2025

Introduction

With GNU Emacs Org-mode and the Python library SymPy it is possible to create calculation sheets, as if you were using pencil and paper. That means, you write formulas and define parameters on a sheet, and let python do the actual calculation. If you change the value of a parameter, the sheet is updated. This it pretty close to an interactive notebook like it is used by programs like MathCAD, but much more flexible[1].

Before we start, we have to enable the execution of python code in Emacs: Therefore, execute `M-x load-library RET ob-python` once[2].

Example: Circumference and Area of a Circle

To give a minimal example, a simple pencil and paper like calculation could look like this:

Given:

$$d_{circ} = 1.250 \cdot \mathrm{m}$$

Formulas:

$$C = \pi \cdot d_{circ}$$

$$A = \frac{\pi \cdot d_{circ}^2}{4}$$

[1] Because you could also use another language like Octave or Ruby and you could export to other formats than pdf, for example html or odt.

[2] As in Emacsc documentation, `M` stands for the meta-key, usually Alt, `C` for the control-key, and `RET` is Return.

Results:

$$C = 3.927 \cdot \mathrm{m}$$

$$A = 1.227 \cdot \mathrm{m}^2$$

Corresponding Org-mode Code

The code in the org-mode document, that generated the above calculation, looks like this:

Given

```
#+BEGIN_SRC python :session :results output latex :exports results
    from sympy import pi
    from sympy.physics.units import meter
    d = symbols('d_{circ}')
    given = { d : 1.25 * meter, }
    prt( given )
#+END_SRC
```

Formulas:

```
#+BEGIN_SRC python :session :results output latex :exports results
    C, A = symbols('C A')
    expr = {}
    expr[C] = d * pi
    expr[A] = d**2 * pi / 4
    prt(expr)
#+END_SRC
```

Results:

```
#+BEGIN_SRC python :session :results output latex :exports results
    res = {}
    res[C] = expr[C].subs(given).evalf()
    res[A] = expr[A].subs(given).evalf()
    prt(res)
#+END_SRC
```

Explanation

Step 1: Define and print expressions

I prefere using dictionaries to store my formulas and parameters, although there are different ways to do this. In the example, the diameter is stored in a dictionary with the

name `given`. Note, that the parameter `d` must be of the type `Symbol`. Additionally, we import the constant `pi` and use physical units.

```
#+BEGIN_SRC python :session :results output latex :exports results
    from sympy import pi
    from sympy.physics.units import meter
    d = symbols('d_{circ}')
    given = { d : 1.25 * meter, }
    prt( given )
#+END_SRC
```

The header of the original org-mode document contains a function `prt`, that uses the `sympy.latex` function for conveniently converting my dictionary to LaTeX Code. You could also use the `latex` function directly, but in this way, it looks a bit nicer and is more convenient. Just for completeness, this is the function definition:

```
#+BEGIN_SRC python :session :exports none
    from sympy import symbols, Symbol, latex, Float, sympify
    # print a dictionary that contains expressions,
    # floats are sympified and then formatted as defined by frmt
    def prt(expr_dict, frmt='{:.4G}'):
        for symbol in expr_dict:
            expr = fmt(sympify(expr_dict[symbol]), frmt)
            print('$$'+ \
                  symbol.name + ' = ' + \
                  latex(expr,
                        mul_symbol    ='dot',
                        imaginary_unit='rj') + \
                  '$$\n'
            )
    # apply format string to floats
    fmt = lambda expr, frmt: expr.xreplace(
        {n: Symbol(frmt.format(n)) for n in expr.atoms(Float)}
    )
    # convert kelvin temperature to a printable celsius expression
    toDegC = lambda elem: (elem.atoms(Float).pop() - 273.15) * symbols(r'\text{degC}')
#+END_SRC
```

Back to the example. Press `C-c C-c` to execute the code. The following `RESULTS` block will be generated automatically:

```
#+RESULTS:
#+begin_export latex
$$d_{circ} = 1.250 \cdot \text{m}$$
#+end_export
```

In the final document, the only visible output will be:

$$d_{circ} = 1.250 \cdot \mathrm{m}$$

In the same way, formulas for circumference C and Area A may be stored in a dictionary named `expr`:

```
#+BEGIN_SRC python :session :results output latex :exports results
    C, A = symbols('C A')
    expr = {}
    expr[C] = d * pi
    expr[A] = d**2 * pi / 4
    prt(expr)
#+END_SRC
```

The ouput is:

$$C = \pi \cdot d_{circ}$$

$$A = \frac{\pi \cdot d_{circ}^2}{4}$$

Step 2 Manipulate and evaluate expressions and print the results

Use standard Sympy functions for manipulating your expressions. For example, to evaluate the above defined formulas with the given parameter d_{circ}, I used the following code:

```
#+BEGIN_SRC python :session :results output latex :exports results
    res = {}
    res[C] = expr[C].subs(given).evalf()
    res[A] = expr[A].subs(given).evalf()
    prt(res)
#+END_SRC
```

This is the output:

$$C = 3.927 \cdot \mathrm{m}$$

$$A = 1.227 \cdot \mathrm{m}^2$$

Code execution

For security reasons, org asks to confirm the execution of each code block. For trusted sources, this behavior can be changed by setting `org-confirm-babel-evaluate` to `nil`. Pressing `C-c C-e l p` will execute all code blocks and export the pen and paper like calculation to a PDF-file.

Quadratic Equation

Just to give another example. Let's define a simple algebraic equation with some parameters, plot a graph and calculate the roots.

Code

Given

```
#+BEGIN_SRC python :session :results output latex :exports results
    f1, a, b, c, x = symbols('f_1(x) a b c x')
    quadr = {
        f1 : a * x**2 + b * x + c,
        a : 1.25,
        b : 2.75,
        c : -3.5,
    }
    prt(quadr)
#+END_SRC
```

Graph

```
#+begin_src python :session :results file :exports results
  from sympy import plot
  fname = './img01.png'
  expr = quatr[f1].subs(quadr)
  plot(expr).save(fname)
  fname
#+end_src
```

Roots

```
#+BEGIN_SRC python :session :results output latex :exports results
  from sympy import solve
  x0, x1 = symbols('x_0 x_1')
  roots = solve(quadr[f1].subs(quadr), x)
  results = {x0: roots[0], x1:roots[1]}
  prt(results)
#+END_SRC
```

Output

This would be, how it looks like in the final document:

Given

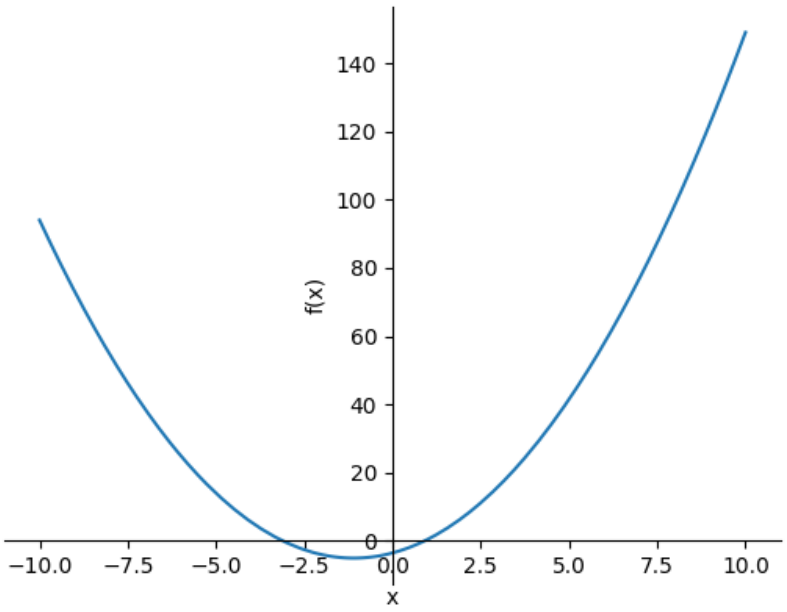
$$f_1(x) = a \cdot x^2 + b \cdot x + c$$

$$a = 1.250$$

$$b = 2.750$$

$$c = -3.500$$

Graph



Roots

$$x_0 = -3.102$$

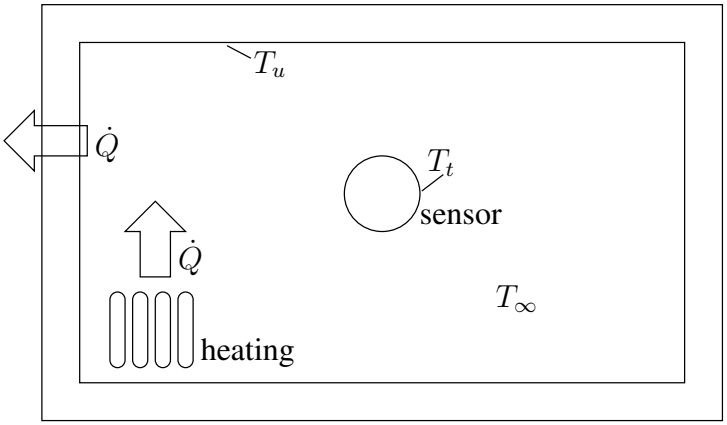
$$x_1 = 0.9025$$

Real World Example

Let's discuss a small heat transfer problem as a first real world example.

Imagine a temperature sensor in a closed room (see figure below). The heat loss through the walls is compensated by a heating, in a way, that temperatures are constant over time. The inner surfaces of the walls have a uniform temperature T_u of 18 °C. The reading of the temperature sensor T_t is 20 °C. The sensor is of type TMP36 with a black plastic casing that has an emissivity ϵ of 0.95. The conductive heat transfer coefficient at the sensor surface α is estimated to be 3 W/(m²·K).

What is the real indoor air temperature T_∞?



Given

Let's list the available information.

$$T_t = 293.2 \cdot \mathrm{K}$$

$$T_u = 291.2 \cdot \mathrm{K}$$

$$\epsilon = 0.9500$$

$$\alpha = \frac{5 \cdot \mathrm{W}}{\mathrm{K} \cdot \mathrm{m}^2}$$

$$c_s = \frac{5.670E-8 \cdot \mathrm{W}}{\mathrm{K}^4 \cdot \mathrm{m}^2}$$

We also included the Stefan-Boltzman constant c_s.

Equations

First, let's understand the problem. The coldest part of the room are the walls. The temperature sensor will radiate heat to the walls. In steady state, the sensor will absorb the same amount of heat from the surrounding air by means of convection. Otherwise, the temperature reading of the sensor would not be steady. So we need two equations. One, describing the radiation from the sensor to the walls $\dot{Q}_r$, and another one for the convection of heat from the air to the sensor $\dot{Q}_c$.

$$Q_r = A \cdot \epsilon \cdot c_s \cdot \left(T_t^4 - T_u^4\right)$$

$$Q_c = A \cdot \alpha \cdot (T_\infty - T_t)$$

Solution

Because $\dot{Q}_r$ and $\dot{Q}_c$ are identical in steady state, we can eliminate them from the equations. The surface area of the thermometer will also vanish. Solving the equation to T_∞ gives:

$$T_\infty = \frac{T_t^4 \cdot \epsilon \cdot c_s + T_t \cdot \alpha - T_u^4 \cdot \epsilon \cdot c_s}{\alpha}$$

Answer

Replacing the symbols by numbers gives:

$$T_\infty = 295.3 \cdot \mathrm{K}$$

$$T_\infty = 22.15 \cdot \mathrm{degC}$$

We could much improve the accuracy of the sensor by wrapping it in a piece of aluminium foil that has an emissivity ϵ of 0.03. A reading of 20 °C would now correspond to the following ambient temperature:

$$T_\infty = 293.2 \cdot \mathrm{K}$$

$$T_\infty = 20.07 \cdot \mathrm{degC}$$

Code

This is the code, that produced the above example. The explanations are omitted.

Given

```
#+BEGIN_SRC python :session :results output latex :exports results
from sympy.physics.units import kelvin, meter, watt
from scipy.constants import sigma
Tt, Ti, Tu, epsilon, alpha, cs = symbols(r'T_t T_\infty T_u \epsilon \alpha c_s')
giv = {
    Tt      : (20+273.15) * kelvin,
    Tu      : (18+273.15) * kelvin,
    epsilon : 0.95,
    alpha   : 5 * watt / (meter**2 * kelvin),
    cs      : sigma * watt / (meter**2 * kelvin**4),
}
prt(giv)
#+END_SRC
```

Equations

```
#+BEGIN_SRC python :session :results output latex :exports results
Qc, A, Qr, = symbols(r'Q_c A Q_r')
eqn = {
    Qr : epsilon * cs * A * (Tt**4 - Tu**4),
    Qc : alpha * A * (Ti - Tt),
}
prt(eqn)
#+END_SRC
```

Solution

```
#+BEGIN_SRC python :session :results output latex :exports results
sol = {
    Ti : solve((eqn[Qr] - eqn[Qc]), Ti)[0]
}
prt(sol)
#+END_SRC
```

Answer

```
#+BEGIN_SRC python :session :results output latex :exports results
ans = {
    Ti : sol[Ti].subs(giv)
}
prt(ans)
prt({Ti:toDegC(ans[Ti])})
#+END_SRC
```

Alternate Answer (Aluminium foil)

```
#+BEGIN_SRC python :session :results output latex :exports results
giv[epsilon] = 0.03
ans = {
    Ti : sol[Ti].subs(giv)
}
prt(ans)
prt({Ti:toDegC(ans[Ti])})
#+END_SRC
```